PARTICLE SWARM OPTIMIZATION

The non-linear optimization models developed in the previous chapter require an optimization method to search for the optimal solutions for each type of riprap stone revetment. The term optimization refers to the procedure of detecting attributes, configurations or parameters of a system that produces the desirable response. In order to apply an optimization procedure, the physical/ engineering problems need to be translated into mathematical models and they must represent all key features of the original system that are under consideration and its accurate simulation. Thus, models offer mathematical means of identifying and modifying the system's properties to produce the most desirable response even without actual construction of the system, thereby they save time and cost as well. The difficulty level in solving an optimization problem relies on the form and mathematical properties of the objective function. It rises sharply up with increasing number of decision variables. The exponential growth of the search space with the problem's dimension is the reason of rise in difficulty level. In such cases, algorithmic procedures that take advantage of modern computing systems can be resorted to, however, only approximate solutions can be obtained under such cases. Thus, computational accuracy, time availability, and implementation efforts become prime determinants while applying computational optimization algorithms.

4.1. Swarm Intelligence

The stream of artificial intelligence that studies the collective behavior and evolving properties of complex, self-organized but decentralized systems with living organism societal structure constitutes swarm intelligence. These systems comprises of simple interacting agents organized in small colonies/ swarms. Each constituent element of the society has a limited action space with no centralized control. The aggregated behaviour of the swarm exhibits fascinating traits of intelligence to respond appropriately towards environmental changes and decision-making capacities. The primary motivation behind the development of swarm intelligence based algorithms stems out from the imitation of the behaviour of societies of living organism in nature. Fish schools, bird flocks, and ant colonies etc amazingly manifest self-organized, collision-free, synchronized and intelligent collective behaviors, which cannot be produced by simply aggregating the behavior of each constituent member. The flight of a bird flock can be imitated with reasonable similarity by maintaining the dynamic distance between a bird and its immediate neighbours. The distance between the two birds varies with the size of a flock. Similarly, an element from a fish school maintains a relatively larger mutual distance while swimming freely. These fish schools (see Fig. 4.1) move together in a close group in the presence of predators. They react to external threats by rapidly changing their form, and breaking in smaller subgroups and re-uniting.

They also demonstrate a remarkable ability to respond collectively to the external stimuli to preserve personal integrity. Despite the physical or structural differences in groups of such species, these living groups possess common properties that are identified as primary attributes of swarm intelligence (Parsopoulos and Vrahatis 2010; Millonas 1994). The first attribute is proximity that refers to the ability of the group members to perform space and time computations. The second is quality that relates to the group ability to respond to environmental quality factors. The third is referred to as diverse response, which is the ability to produce a plurality of different responses.

The fourth is termed as stability, which ensures the group members ability to retain robust behaviors under mild environmental changes, and finally the fifth is adaptability that reflects the ability to change behaviour when it is dictated by external factors.

In mid-90's a different category of algorithms under the name of swarm intelligence appeared. They replicate the societal structure and aggregating behaviour of organized colonies of simple organisms such as ants, bees, and fish. These search algorithms mimic the behaviour of populations of species that interact for their livelihood and/or survival. The constituent elements of these societies display a limited range of their individual responses but their society as a whole exhibit a fascinating behavior with identifiable traits of intelligence. The societal dynamics of these algorithms were modeled by the mathematical equations involving stochastic processes. One such swarm intelligence based search algorithm is particle swarm optimization, which is applied for solving the non-linear optimization problem in the present work.

4.2 Early Precursor of Particle Swarm Optimization

The early version of PSO was developed by Russell C. Eberhart (Purdue School of Engineering and Technology, Purdue University, Indianapolis) and James Kennedy (Bureau of Labor Statistics, Washington, DC) as simulators of social behaviour of bird flocks. The primary rules employed by them to produce swarming behavior were matching the velocities of nearest neighbours and acceleration by distance in their search of food. Having realized the potential of their simulation models, Eberhart and Kennedy refined their model and published the first version of PSO in 1995 (Eberhart & Kennedy, 1995; Kennedy & Eberhart, 1995).

For expressing the optimization problem in a mathematical framework, say $A \in \mathbf{R}^{n}$ be the search space, and $f\colon A \rightarrow Y \subseteq \mathbf{R}$ be the objective function. To keep the problem rather simple, it can be presumed that A is a feasible space of the problem with no further explicit constraints posed on the solution members. No additional assumptions are also imposed regarding the form of the objective function and search space. The algorithm employs a population of feasible solutions that moves stochastically in the search space to explore the search space. The population is referred to as the swarm and its elements are called as the particles. The swarm is defined as a set:

$S = \{x_1, x_2, ...x_N\}$ of *N* particles

where a particle is defined as:

$x_i = (x_{i1}, x_{i2}, ..., x_{in})^T \in A$, i=1, 2, ….., N

Indices are arbitrarily assigned to particles, while *N* is a user-defined parameter of the algorithm and it shows the number of particles in a swarm. The objective function *f*(*x*) is assumed to be available for all points in *A*. Therefore, each particle has a unique function value, $f_i = f(x_i) \subseteq Y$ and they move within the search space (*A*) iteratively. This is achieved by adjusting the position of particles by a proper position shift referred to as velocity. It is denoted as:

$v_i = (v_{i1}, v_{i2}, ..., v_{in})^T$ i=1, 2,….., N

The velocities of the particles are also adapted iteratively to enable them capable of potentially visiting any region of *A*. If *t* represents the iteration counter, then the current position of the *i*-th particle and its velocity will henceforth be denoted as $x_i(t)$ and $v_i(t)$, respectively. The velocity of a particle is updated according to the information obtained during the previous iteration of the algorithm. This is implemented in terms of a memory, where each particle can store the best position it has ever visited during its search. For this purpose, besides the swarm (S) which contains the current positions of the particles, PSO also maintains a memory set defined as:

$$P = \{p_1, p_2, \ldots\ldots\ldots, p_N\}$$

It contains the best positions of the particles and is given as:

$p_i = (p_{i1}, p_{i2}, \ldots\ldots, p_{in})^T \in A$ for $i = 1, 2, \ldots, N$ ever visited by each particle.

These positions are defined as:

$$p_i(t) = \arg\min_t f_i(t)$$

where *t* stands for the iteration counter.

Since the PSO is based on simulation models of social behavior, therefore, an information exchange mechanism will exist between the particles that allows them mutually communicate their experience. The algorithm approximates the global minimizer with the best position ever visited by all particles, therefore, this crucial information needs to be shared among all the particles. Let *g* be the index of the best global position with the minimum function value in ***P*** at a given iteration *t*,

$$p_g(t) = \arg\min_t f\left(p_i(t)\right)$$

Then, the early version of PSO is described by the following mathematical equations (Eberhart & Kennedy, 1995; Eberhart *et al.*, 1996; Kennedy & Eberhart, 1995):

$$v_i(t+1) = v_i(t) + c_1 r_1\left(p_i(t) - x_i(t)\right) + c_2 r_2 (p_g(t) - x_i(t)) \qquad (4.1)$$

$$x_i(t+1) = x_i(t) + v_i(t+1) \; \forall \; i = 1, 2, \ldots, N \qquad (4.2)$$

where t denotes the iteration counter. r_1 and r_2 are uniformly distributed random variables within [0,1]; and c_1 and c_2 are weighting factors referred to as the cognitive and social parameter, respectively.

In the first version of PSO, a single weight, $c = c_1 = c_2$, which was called as acceleration constant was used instead of the two distinct weights in equation (4.1). However, it was different in later version of PSO that offered a better control on the algorithm. Best positions (memory) are updated at each iteration after the update and evaluation of particles. Thus, the new best position of x_i at iteration t+1 is defined as follows:

$$p_i(t+1) = \begin{cases} x_i(t+1) & if \quad f(x_i(t+1)) \leq f(p_i(t)) \\ p_i(t) & otherwise \end{cases} \tag{4.3}$$

The new determination of index *g* for the updated best positions completes an iteration of PSO. The operational steps of PSO are shown below:

Step 1. Set $t \leftarrow 0$

Step 2. Initialize ***S*** and Set ***P*** ≡ ***S***

Step 3. Evaluate ***S*** and ***P***, and define index *g* of the best position

Step 4. While (termination criterion not met)

Step 5. Update ***S*** using equations (4.1) and (4.2)

Step 6. Evaluate ***S***

Step 7. Update ***P*** and redefine index *g*

Step 8. Set $t \leftarrow t+1$

Step 9. End of loop

Step 10. Print best position found.

Particles of the swarm are initialized randomly over the search space (A) by following a uniform distribution. The choice of uniform distribution treats every region of A equally capable of having global minimum. Therefore, it is preferred for all cases where there is no information on the form of the search space or the objective function are available. Since all modern computer systems are equipped with a uniform random number generator, therefore, it can be implemented easily. The previous velocity term $v_i(t)$ in the right-hand side of equation (4.1) provides a means of inertial movement to the particle by accounting its previous position shift into consideration. This property can prevent it from becoming biased towards the involved best positions that may entrap it into local minima. Furthermore, the previous velocity term serves as a perturbation for the global best particle, x_g. In fact, if a particle x_i discovers a new position with lower function value than the best one, it becomes the global best (i.e., $g \leftarrow i$). Its best position p_i in the next iteration will coincide with p_g and x_i. Thus, the two stochastic terms in equation (4.1) will vanish. If there had been no previous velocity term in equation (4.1), the aforementioned particle would stay at the same position for several iterations, until a new best position is detected by another particle. The velocity term thus allows this particle to continue its search, following its previous position shift. The values of c_1 and c_2 affect the search ability of PSO by biasing the sampled new positions of a particle x_i towards the best positions p_i and p_g, respectively, as well as by changing the magnitude of search. If a better global exploration is required, then high values of c_1 and c_2 provide new points in relatively distant regions of the search space. On the other hand, a more refined local search around the best positions achieved so far would require the selection of smaller values for the two parameters c_1 and c_2. A relatively higher value of c_1 than c_2 would cause the sample to be bias towards the direction of p_i, while in opposite case, i.e., c_1 is less than c_2 would shift the population towards the direction of p_g. This feature can be utilized in cases where there is special information is available regarding the form of the objective function. For example, a choice of c_1 less than c_2 promotes sampling closer to p_g, which will make the optimization algorithm to be more efficient for the problem having convex unimodal objective function, provided it is combined with a proper search magnitude.

In most of the optimization problems, it is very essential that the particles always lye within the feasible search space. To ensure this, bounds are imposed on the position of each particle x_i to restrict it within the search space. If a particle acquires a value out of the search space after the application of equation (4.2), it is immediately dragged to its boundary. In a simple case, where the search space is defined as a box:

$$A = [a_1\, b_1] \times [a_2\, b_2] \times [a_3\, b_3] \times \ldots\ldots \times [a_n\, b_n] \tag{4.4}$$

with a_j, $b_j \in \mathbf{R}$, j = 1, 2,…, n, the particles are restricted as follows:

$$x_i(t+1) = \begin{Bmatrix} a_j & if & x_i(t+1) < a_j \\ b_j & if & x_i(t+1) > b_j \end{Bmatrix} \quad i=1, 2, \ldots N \qquad (4.5)$$

Alternatively, a bouncing movement off the boundary back into the search space has been considered,similarly to a ball bounced off a wall (Kao *et al.*, 2007). This approach is less popular because it requires the modeling of particle motion with complex physical equations. For the cases where *A* cannot be described as a box, special problem-dependent conditions may be necessary to restrict particles going out of the search domain.

Early PSO variants performed reasonably well for simple optimization problem but their crucial deficiencies were revealed when they were applied on relatively harder problems with a larger search spaces and multitude of local minima. Thus, early version of particle swarm optimization required refinements to address its deficiencies.

4.3 Swarm Explosion and Velocity Clamping

The first significant issue related to the swarm explosion effect that referred to the uncontrolled increase in the magnitude of the velocities, which resulted in swarm divergence. Lack of a velocity control mechanism in early particle swarm optimization was the root of this deficiency. It is addressed by using strict bounds for velocity clamping at desirable levels that prevents particles from taking extremely large steps from their current position. A user-defined maximum velocity threshold $v_{max} > 0$ is normally considered. After determining the new velocity of each particle by equation (4.1), the following restrictions are applied prior to the position update with equation (4.2):

$|v_i(t+1)| \leq v_{max}$ for $i = 1, 2, \ldots, N$

In case of violation, the velocity component of a particle is set directly to the closest velocity bound as:

$$v_i(t+1) = \begin{Bmatrix} v_{\max} & if & v_i(t+1) > v_{\max} \\ -v_{\max} & if & v_i(t+1) < -v_{\max} \end{Bmatrix} \tag{4.6}$$

However, different velocity bounds as per the direction component can also be used. The value of v_{max} is usually taken as a fraction of the search space size per direction. If the search space is defined as given in equation (4.4), a common maximum velocity for all directions can be defined as:

$$v_{\max} = \frac{\min_i \{b_i - a_i\}}{k} \tag{4.7}$$

where the value of $k = 2$ is usually adopted. In case, a problem requires smaller particle steps, a relatively higher value of k is adopted. However, if the search space has a multitude of minimizers with narrow regions of promising zones close to each other, the magnitude of k should assume adequately large value to prevent particles from overflying the several attractive zones. At the same time, *k* should not take a very small or large value that hampers the satisfactory exploration of the search space.

The velocity clamping provision offered an efficient solution to the problem of swarm explosion but it did not address the problem of convergence. The particles are now able to fluctuate around their best positions but they were unable to achieve convergence on a promising zone/position or perform a further refined search around the promising zone. This issue was resolved by the introduction of a new parameter in the original particle swarm optimization model as described below.

4.4 Concept of Inertia Weight

The application of a maximum velocity threshold improved the performance of early particle swarm optimization but it could not render the algorithm efficient for complex optimization problems. Despite the elimination of swarm explosion, the swarm could not concentrate its particles around the most promising zones in the last phase of the optimization procedure. Thus, no further refinement in solutions could be obtained even after the detection of promising zone of the search space, which, in turn, causes particles to oscillating on wide trajectories around their best positions. The reason for this deficiency was associated to the disability of the algorithm to control the velocities. A refined search in promising zones requires a strong attraction of the particles towards them and small position shifts that prohibit particles to escape from close vicinity. This is possible by reducing the perturbations that shift particles away from best positions. Therefore, the effect of previous velocity on the current iteration velocity needs to fade away with iterations for each particle. Thus, a new parameter (w) referred to as inertia weight was introduced in equation (4.1), and the resulting version of particle swarm optimization (Eberhart & Shi, 1998; Shi & Eberhart, 1998a; Shi & Eberhart, 1998b) takes the following form:

$$v_i(t+1) = wv_i(t) + c_1 r_1\big(p_i(t) - x_i(t)\big) + c_2 r_2 (p_g(t) - x_i(t)) \qquad (4.8)$$

$$x_i(t+1) = x_i(t) + v_i(t+1) \;\forall\; i = 1, 2, \ldots, N \qquad (4.9)$$

The rest of the parameters remain the same as for the early particle swarm optimization variant shown by equations (4.1) and (4.2). The value of the inertia weight will be selected in such a manner that the effect of $v_i(t)$ fades away during the execution of the algorithm. Thus, a decreasing value of w with time is preferable. Therefore, a common practice is to initialize w by a value slightly greater than 1.0 (say, 1.2) to promote exploration in early stages with gradual reduction towards the value of zero in later stages to eliminate oscillatory behaviour. A strictly positive lower bound on w (e.g., 0.1) is used to prevent the previous velocity term from vanishing. A linearly decreasing scheme for w can be described as:

$$w(t) = w_{up} - (w_{up} - w_{low})\frac{t}{T} \tag{4.10}$$

where t stands for the iteration counter; w_{low} and w_{up} are the desirable lower and upper bounds of w; and T is the total allowable number of iterations. Equation (4.10) produces a linearly decreasing time-dependent inertia weight with starting value w_{up} at iteration $t = 0$, and final value w_{low} at the last iteration $t = T$.

The amazing improvement in the performance of particle swarm optimization gained by using the inertia weight with velocity clamping (see Figure 4.1), rendered it to become the most popular particle swarm optimization method. Although particles were able to avoid swarm explosion and converge around the best positions but they were still getting easily trapped in local minima, especially in complex problems.

4.5 Standard Particle Swarm Optimization

The efficiency of particle swarm optimization variants attracted the interest of the scientific community. Ozcan and Mohan (1999) published the first theoretical investigation of particle swarm optimization for multi-dimensional space problem and provided closed-form equations for particle trajectories. Their study focused on the early equations (4.1) and (4.2) of particle swarm optimization, and they showed that the particles were actually moving on sinusoidal waves per coordinate of the search space, while stochasticity offered a means to manipulate its frequency and amplitude. A few years later, this interesting result was followed by a thorough investigation by Clerc and Kennedy (2002), who considered different generalized particle swarm optimization models to perform a dynamic system analysis of their convergence. They offered a solid theoretical background to the algorithm, and established one of the investigated models as the default contemporary particle swarm optimization variant. This model is expressed by the following equations:

$$v_i(t+1) = \chi\left[v_i(t) + c_1 r_1\left(p_i(t) - x_i(t)\right) + c_2 r_2 (p_g(t) - x_i(t))\right] \tag{4.11}$$

$$x_i(t+1) = x_i(t) + v_i(t+1) \ \forall \ i = 1, 2, \ldots, N \tag{4.12}$$

where χ is referred to as constriction coefficient or constriction factor. It is distinguished in literature due to its theoretical properties that imply the following explicit relation among its parameters (Clerc & Kennedy, 2002):

$$\chi = \frac{2}{\psi - 2 - \sqrt{\psi^2 - 4\psi}} \tag{4.13}$$

where $\psi = c_1 + c_2 > 4$

Based on this equation, the setting: χ = 0.729, $c_1 = c_2$ = 2.05, is usually viewed as the default parameter setting of the constriction coefficient particle swarm optimization variant.

The stochastic parameters, r_1 and r_2, are considered to be uniformly distributed within the range [0, 1]. Alternatively, r_1 and r_2 can be uniformly distributed within a sphere centered at the origin, with radius equal to 1. A different approach suggests that r_1 and r_2 can be normally distributed with a Gaussian distribution, $\boldsymbol{N}(\mu,\sigma^2)$. In this case, the distribution of new prospective positions significantly differ from the previous two cases, depending heavily on the values of μ and σ, which, in turn, are usually dependent on the parameters c_1 and c_2. The aforesaid distribution was applied in the vast majority of optimization works with the rectangular one being the most popular. Application of different distributions are also available but with less frequency. Although they have shown potential for enhanced efficiency in specific problems, these approaches cannot be characterized as standard particle swarm optimization variants. The case of rectangular distributions offers a simple and efficient default choice. The case of spherical distributions is quite similar to the rectangular one but it restricts the range of possible new particle positions. On the other hand, the use of Gaussian distributions offers a completely different capability to the algorithm. Depending on the values of parameters, a particle can move in a direction opposite to that of the best positions, which exhibits a completely different dynamics than the standard particle swarm optimization models. The efficiency of such model always depends on the problem at hand. If the particle swarm optimization has detected the most promising zones of the search space, the application of Gaussian distribution can slow down the convergence rate. If there are several local minima with narrow basins of attraction, located closely to the global minimum, this approach works effectively for the algorithm. Since the type of distribution applied in the problem has significant effect on performance, therefore, a careful selection is essential for the convergence of the swarm to the global minimum.

Fig. 4.1 A fish school dividing into two groups for their survival in the presence of a predator

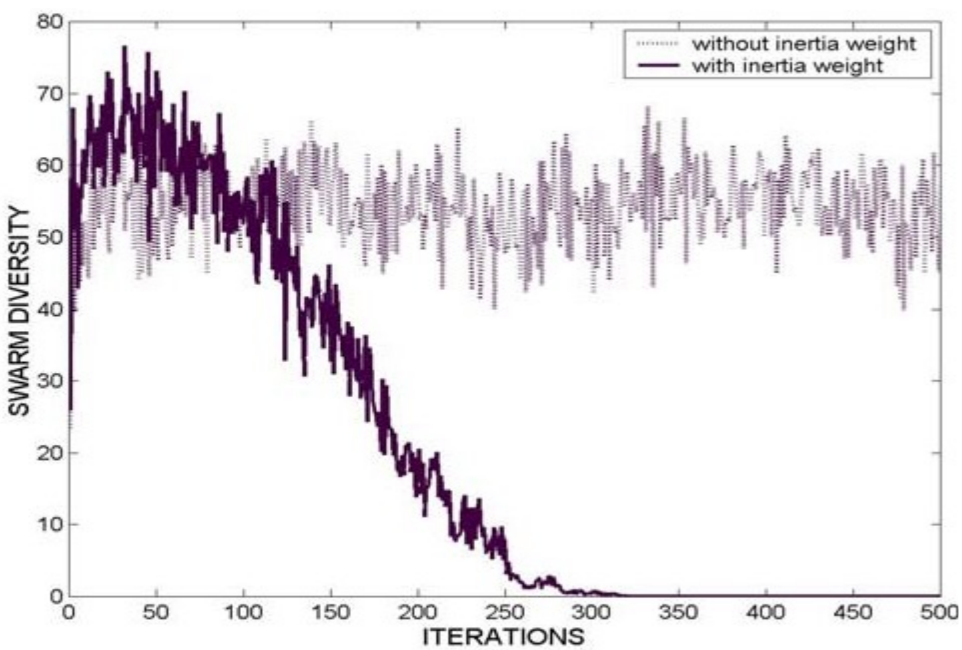

Fig. 4.2 Swarm diversity during search with (solid line) and without (dotted line) inertia weight (Adopted from Parsopoulos and Vrahatis 2010)